Managing Kernel Extensions

Session 501

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Managing Kernel Extensions

Nik Gervae, Andrew Myrick
I/O Kit Team Engineers

Only on Mac OS

Do You Need to Be Here?

Reasons not to write a kernel extension

- Kernel code runs without protections of user-space programs
- A mistake can bring the whole OS down
- Debugging kernel code is harder
- Code does not run faster in the kernel

- Mac OS X makes possible many things from user space
  - Printer, scanner, and camera drivers
  - Many kernel services, including several device classes
  - See the user-space IOKit.framework
- Check user-space facilities before resorting to kernel coding

Do You Need to Be Here?

Things that require a kernel extension

- Low-level device drivers
  - Need register access to hardware
  - Need to handle interrupts
  - Clients in the kernel
- Network stack filters
- Kernel authorization plug-ins
- File systems

What You'll Learn

General info

- Structure of a kernel extension (kext)
- Use of kext tools
- Installing kexts
- Debugging kexts
- How the kext system works
- Kext runtime support

What You'll Learn

Major new features in Snow Leopard

- 64-bit kexts
- Architecture-specific properties
- kextload → kextutil
- Programmatic kext loading
- Resource access from kernel

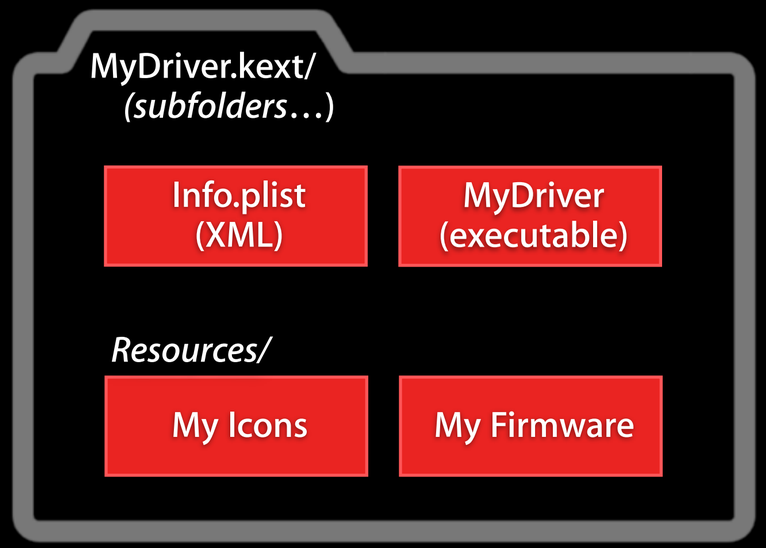
What You Won't Learn

- Kernel programming details
- Specific programming interfaces
  - Drivers
  - Network filters
  - Kernel authorization plug-ins
  - File systems

Kernel Extension Fundamentals

Structure, types, and tools

What Is a Kernel Extension?



All files and folders owned by root:wheel and nonwritable

- A plug-in for the Mac OS X kernel
- Packaged as a bundle
  - Info.plist file describes the kext (XML format only; no comments)
  - Executable loads and runs in kernel
  - One level of plug-in kexts
- Xcode templates
  - IOKit Driver
  - Generic Kernel Extension

Info.plist Properties

- Core properties
  - CFBundleIdentifier, CFBundleExecutable
  - CFBundleVersion (0.0.0d1–9999.99.99)
  - OSBundleLibraries
- Additional properties
  - OSBundleRequired
  - OSBundleCompatibleVersion
  - OSBundleAllowUserLoad

  - OSBundleEnableKextLogging

  - IOKitPersonalities

Architecture-Specific Properties

- Add *_arch* to a top-level OS… or IO… key
  - OSBundleLibraries_x86_64
  - IOKitPersonalities_x86_64
- Allows linking or driver matching based on kernel architecture
- Keep the base key for backward compatibility
- Specific to kexts—not a CFBundle enhancement

Kext Executable

- A (multiarchitecture) Mach-O file
- Embeds bookkeeping info
- Codeless kext—no executable
  - Publish IOKitPersonalities naming another kext
  - Redirect linkage to another library

Kext Resource Files

- Any data relevant to bundle
  - Icons
  - Other localized files (user space only)
  - Firmware
  - Lookup tables
- Now accessible from kernel task
- Part of your kext bundle
  - Read-only, static data
  - Must be immutable
  - Do not use for configuration/preferences

XCode Kext Templates

	IOKit Driver	Generic Kext
Language	C++	C
Implementation	Subclasses	Registered C functions
Initialization and teardown	C++ statics and instance functions	Kext start/stop functions
Loading	Automatically by I/O Kit	Explicitly
Unloading	Automatically by C++ instance count	Explicitly

Kext Tools

User-space programs that work on kexts

Development	Deployment	System
kextutil	kextload	kextd
kextlibs	kextunload	kextcache
kextstat		
kextfind		

Loading and Unloading Kexts

Tools and programming interfaces

Deployment Loading: kextload

- A significant change in Snow Leopard
- Insulates installation/deployment from development
- Sends an RPC to kextd, which loads into the kernel
- Simple arguments/options
  - Kext bundle names
  - -bundle-id (-b) *identifier*
  - -repository (-r) *folder*
  - -dependency (-d) *kext*
  - -verbose (-v), -quiet (-q)

Development Loading: kextutil

- *Not* for deployment loading
- Requires root access to load into the kernel
  - Talks directly to the kernel
- Moves developer options from kextload for Snow Leopard
  - Sanity checking
  - Staged loading to debug kext startup and matching
  - Generating debug symbols
  - Cross-architecture

Unloading: kextunload

- Primarily for development and (un)installers
- Requires root access
- Shuts down kext, unloads, and removes driver matching info
- Unload by bundle filename or identifier

Programmatic Loading from User Space

```
#include <IOKit/kext/KextManager.h>
#include <libkern/OSKextLib.h>

OSReturn KextManagerLoadKextWithURL(
  CFURLRef kextURL,
  CFArrayRef dependencyAndRepositoryURLs);
```

```
OSReturn KextManagerLoadKextWithIdentifier(
  CFStringRef kextIdentifier,
  CFArrayRef dependencyAndRepositoryURLs);
```

- RPC to kextd (now used by kextload)

Kexts Loadable by Non-Root Users

- OSBundleAllowUserLoad boolean property
- Kext and all dependencies must set this property
- Kext and all dependencies must be in /System/Library/Extensions/ or /System/Library/Filesystems/
- Available for kextload and KextManager APIs
- Not available for kextutil

- Intended for generic kexts
- Unload still requires root privilege
- Use with care

Debugging Kexts

Troubleshooting, debugger setup, and symbol generation

Kext Debugging Techniques

	kextutil -nt	Kext Diagnostics (-verbose, etc.)	IOKitDebug	Log from Kext	Debugger (gdb)
Kext fails to load /start	✓	✓		✓	✓
Driver fails to match			✓	✓	✓
Kext misbehaves				✓	✓
Panic					✓

Sanity Checking with kextutil

```
kextutil -nt -arch x86_64 MyKext.kext
```

- First check to make when your kext fails to load
- -no-load (-n) performs most checks and prints diagnostics
  - Does trial link to report undefined symbols
  - No longer authenticates ownership and permissions
  - Add -print-diagnostics (-t) for authentication
- Uses running kernel arch by default
- Use -arch if checking for a different arch

Kext Tool Diagnostics

- Kext tools' -verbose (-v) flag now offers more control
- kextutil focuses log messages
  - Only kexts being loaded
  - No more spew about all kexts—unless you ask for it
- Tools now capture both user- and kernel-space log messages
- See kext_logging(5) man page

Kext System Diagnostics: Kernel and kextd

- Kernel logging now goes to /var/log/kernel.log
- kextd logging goes to /var/log/system.log

- Kext load logging now available for kernel and kextd
- See kext_logging(5) man page

Kernel Debugging with gdb

Typical setup

Development

Developer tools
Kernel Debug Kit for test release

Kext source
Unstripped kext (Xcode)
Kext dSYM file (Xcode)
Kext debug symbol file (kextutil)

Ethernet

Test

Computer started with boot-args "debug" set:
`sudo nvram boot-args="debug=0x14e -v"`

Connected to Ethernet
Kext loaded in kernel
Panic or NMI (power button)

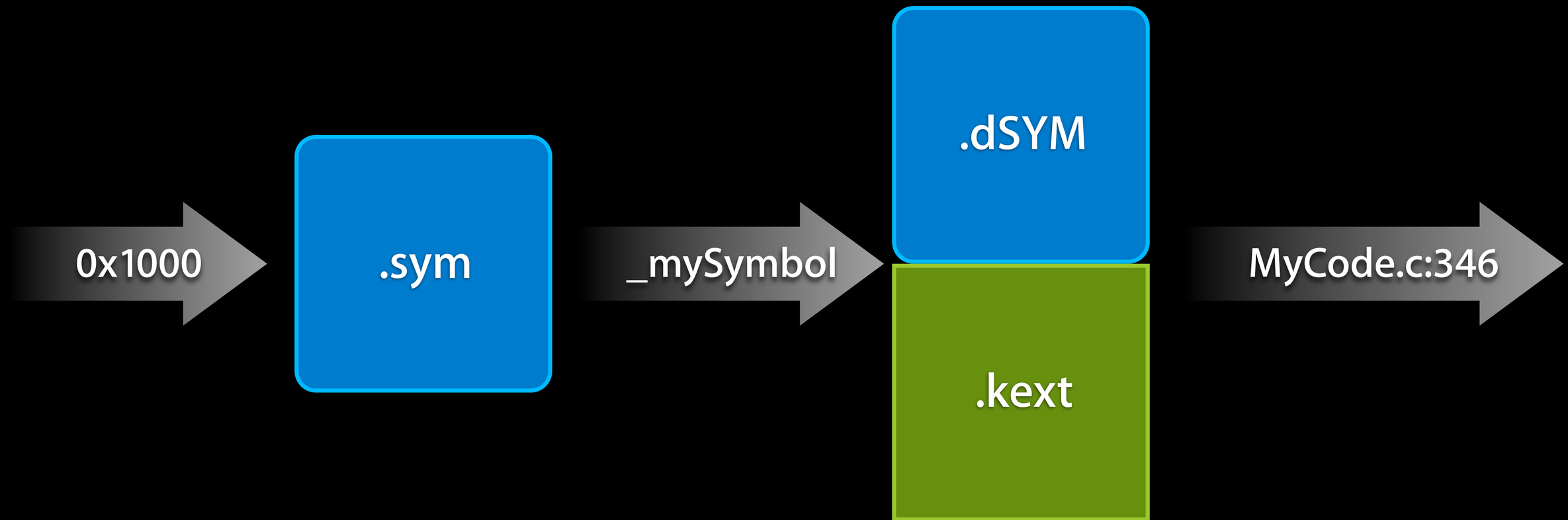
Kernel Debug Kit

- Download from ADC member site
- Contains symboled kernel and Apple library kexts
- One per Mac OS X release (and software updates)
- Now includes debug build of kernel and System.kext

  - See readme file for instructions

Kext Debug Files

From address to source code



Kexts and gdb

Starting gdb for kext debugging

```
% gdb -arch i386 /Volumes/KernelDebugKit/mach_kernel
(gdb) target remote-kdp
(gdb) attach 192.168.1.1
(gdb) source /Volumes/KernelDebugKit/kgmacros
(gdb) set kext-symbol-file-path /tmp
```

- remote-kdp sets up remote kernel debugging
- attach connects to test computer
- kgmacros provides many handy shortcuts
- kext-symbol-file-path tells gdb where to find kext symbol files
- Need to generate symbol files now

Generating Debug Symbols

When loading kext

```
sudo kextutil -s /tmp /tmp/MyKext.kext
```

- Loads MyKext.kext
- Sends IOKitPersonalities to kernel for driver matching
- `-s` saves kext and library .sym files in named folder

- Good for nonpanic problems after kext startup
- Copy .sym files to development computer

Generating Debug Symbols

Kext loaded on same architecture and release

```
kextutil -s /tmp -n /tmp/MyDriver.kext
Enter the hexadecimal load addresses for these extensions
(press Return to skip symbol generation for an extension):
com.apple.iokit.IOUSBFamily:
com.mycompany.mydriver: 0x4756000
```

- -n does not load kext or send IOKitPersonalities
- Prompts for load addresses of dependencies and MyDriver.kext
- Skip dependencies you don’t need
- Add -r /tmp for dependency resolution in /tmp/
- Good for any kext already loaded on test computer

Getting Kext Load Addresses

```
(gdb) source /Volumes/KernelDebugKit/kgmacros
(gdb) showallkmods
kmod         address     size        id    refs     version  name
0x340dca10   0x3317d000  0x0000d000  102      0      1.0d34  com.mycompany.mykext
0x350a6ae0   0x3509c000  0x0000b000  101      0        69.0  com.apple.iokit.IOSurface
0x34fec960   0x34fe8000  0x00005000  100      0     1.9.2d0  com.apple.driver.AppleHWSensor
...
```

Getting Kext Load Addresses

```
% kextstat
Index Refs Address            Size       Wired      Name (Version) <Linked Against>
    1   57 0                  0          0          com.apple.kpi.bsd (10.0.0d8)
    3   78 0                  0          0          com.apple.kpi.iokit (10.0.0d8)
    4   83 0                  0          0          com.apple.kpi.libkern (10.0.0d8)
    5   74 0                  0          0          com.apple.kpi.mach (10.0.0d8)
...
  101    0 0x3509c000 0xb000     0xa000     com.apple.iokit.IOSurface (69.0) <7 5 4 3 1>
  102    0 0x3317d000 0xd000     0xc000     com.mycompany.mykext (1.0d34) <5 4 1>
...
```

Generating Debug Symbols

Kext loaded on different architecture, same release

```
kextutil -s /tmp -n
  -arch i386
  -k /Volumes/KernelDebugKit/mach_kernel
  /tmp/MyDriver.kext
```

- -arch saves symbols for specified arch
- For running kernel arch, kextutil uses running kernel for link info
- For different arch, -k (-kernel) indicates kernel file for link info

- Note: K64 is a different arch (x86_64) from K32 (i386)

Generating Debug Symbols

Kext loaded on different architecture, different release

```
kextutil -s /tmp -n
  -arch i386
  -k /Volumes/KernelDebugKit/mach_kernel
  -e
  -r /Volumes/KernelDebugKit
  /tmp/MyDriver.kext
```

- -e skips /System/Library/Extensions/ on running release
- Use -r for library kexts of other release
- Use Kernel Debug Kit for appropriate release!

Kexts and gdb

Adding kext symbols

```
(gdb) add-kext /tmp/MyKext.kext
```

- add-kext looks for three kext files in same folder
  - MyKext.kext
  - MyKext.kext.dSYM
  - com.mycompany.mykext.sym (bundle ID + .sym)
- Also looks in kext-symbol-file-path for .sym files
- Now set breakpoints and debug

Debugging Kext Startup

A chicken-and-egg problem

- Debugging requires load address…
- So kext must be loaded…
- But loading starts kext—or does it?

- kextutil to the rescue
  - Pause before starting kext
  - Pause before starting matching
  - Delays auto-unload by at least 2 minutes
- For drivers: Unload kext first to remove IOKitPersonalities

Debugging Kext Startup

Interactive startup and driver matching

```
% sudo kextutil -i -s /tmp /tmp/MyDriver.kext
Load /tmp/MyDriver.kext and its dependencies into the kernel [Y/n]? y
Loading /tmp/MyDriver.kext.
/tmp/MyDriver.kext successfully loaded (or already loaded).

/tmp/MyDriver.kext and its dependencies are now loaded, but not
started (unless they were already running). You may now set
breakpoints in the debugger before starting them.
Start /tmp/MyDriver.kext [Y/n]? y
/tmp/MyDriver.kext started.
Send personalities for /tmp/MyDriver.kext [Y/n]? y
Send personality MyDriver Ultra [Y/n]? y
```

…and so on

Debugging Kext Startup

Driver startup (alternate technique)

```
sudo kextutil -l -s /tmp /tmp/MyDriver.kext
```

- -l loads MyDriver.kext without doing driver matching
- Driver delays auto-unload for at least 2 minutes
- Copy .sym files to development computer
- Set breakpoints for driver `::probe()` and/or `::start()`

```
sudo kextutil -m -s /tmp /tmp/MyDriver.kext
```

- -m sends IOKitPersonalities to kernel to start driver matching

Generating Debug Symbols

Other options

- Using load address in running kernel with -A (-use-load-addresses)
- Identifier and load address on command line with -a (-address)
- Interactive for libraries with -I (capital ‘i’, or -interactive-all)
- See kextutil(8) man page

More on Kernel Debugging

- Kernel Extension Programming Topics
- Kernel Programming Guide
- Technical Note TN2063: Understanding and Debugging Kernel Panics
- Technical Note TN2118: Kernel Core Dumps

Kext Dependencies

Libraries for Kernel Programming

Andrew Myrick
I/O Kit Team Engineer

Types of Dependencies

- Built-in kernel programming interfaces (KPIs)
  - Export kernel subsystem symbols
- Library kexts (I/O Kit families)
  - Define and export C++ superclasses

- Kernel.framework/Headers/—all headers for Apple KPIs and libraries

Kernel Programming Interfaces

Four public KPIs

com.apple.kpi.mach	com.apple.kpi.libkern
Time Threads Locks	Memory and strings Atomic operations Utility C++ classes

com.apple.kpi.bsd	com.apple.kpi.iokit
Networking File systems kauth	Core I/O Kit classes

com.apple.kpi.unsupported
Avoid File bug for supported interface

Kernel Compatibility Interfaces

- All interfaces of the form com.apple.kernel*
- Provided only for compatibility with 10.3.9 or earlier
- Available only for 32-bit kernel
- Do not use with any com.apple.kpi.* library

Multiarchitecture Kexts

- One kext can be 32-bit and 64-bit
- Available kernel interfaces vary
- Use architecture-specific OSBundleLibraries property if needed

	32-bit	64-bit
com.apple.kpi.*	✓	✓
com.apple.kpi.unsupported	–	–
com.apple.kernel*	–	✗

OSBundleLibraries

- Lists load time kext dependencies
  - Not build time
- Keys: Library kext identifiers
- Values: Minimum required library versions
- Example: HelloIOKit.kext

```
<key>OSBundleLibraries</key>
<dict>
        <key>com.apple.kpi.iokit</key>
        <string>10.0b1</string>
        <key>com.apple.kpi.libkern</key>
        <string>10.0b1</string>
</dict>
```

OSBundleLibraries

Arch-specific properties

- HelloIOKit.kext with an extra dependency for 64-bit

```
<key>OSBundleLibraries_i386</key>
<dict>
        <key>com.apple.kpi.iokit</key>
        <string>8.0</string>
        <key>com.apple.kpi.libkern</key>
        <string>8.0</string>
</dict>
<key>OSBundleLibraries_x86_64</key>
<dict>
        <key>com.apple.kpi.iokit</key>
        <string>10.0b1</string>
        <key>com.apple.kpi.libkern</key>
        <string>10.0b1</string>
        <key>com.apple.kpi.mach</key>
        <string>10.0b1</string>
</dict>
```

Generating OSBundleLibraries: kextlibs

- Determines the OSBundleLibraries needed by a kext
  - Based on the current running system
  - Run on the oldest target release (10.5 or later)
- Use -xml to generate raw XML

```
$ kextlibs -xml HelloIOKit.kext
<key>OSBundleLibraries</key>
<dict>
        <key>com.apple.kpi.iokit</key>
        <string>10.0b1</string>
        <key>com.apple.kpi.libkern</key>
        <string>10.0b1</string>
</dict>
```

Generating OSBundleLibraries: kextlibs

Arch-specific dependencies

```
$ kextlibs -xml HelloIOKitTwo.kext
<key>OSBundleLibraries_i386</key>
<dict>
        <key>com.apple.kpi.iokit</key>
        <string>10.0b1</string>
        <key>com.apple.kpi.libkern</key>
        <string>10.0b1</string>
</dict>
<key>OSBundleLibraries_x86_64</key>
<dict>
        <key>com.apple.kpi.iokit</key>
        <string>10.0b1</string>
        <key>com.apple.kpi.libkern</key>
        <string>10.0b1</string>
        <key>com.apple.kpi.mach</key>
        <string>10.0b1</string>
</dict>
```

kextlibs Known Issues

- -c (-compatible-versions) is reversed
  - Compatible versions by default
  - Current versions when -c is specified
- -r (-repository) is ignored
  - Only looks in /System/Library/Extensions/

Finding Kexts on Disk: kextfind

- Like find(1) for kexts
- Alternative to kextlibs when targeting any release
  - -dsym finds all kexts that define a symbol
  - Use -f to specify older release's /System/Library/Extensions/

```
$ sudo kextutil /tmp/HelloIOKitBroken.kext/
(kernel) kxld[com.company.driver.HelloIOKit]: The following symbols are unresolved:
(kernel) kxld[com.company.driver.HelloIOKit]: _clock_get_uptime

$ kextfind -dsym _clock_get_uptime
/System/Library/Extensions/System.kext/PlugIns/Mach.kext
```

Kext Linking with kxld

kxld

The kext linker

- Links kext executables into the kernel
  - Resolves undefined symbols
  - Ensures binary compatibility
- Built from scratch for Snow Leopard
  - Performance
  - Portability
  - Extensibility

Kext Linker Architecture

- Intermediate state objects (“link state”)
  - For library kexts
  - Exported symbols and vtables
  - Pageable memory
- Cross-architecture
  - 64-bit capable
  - Byte-order independent
- Kernel and user space

Linking a Kext

.kext

Linked Kext

Linked Kext

Linked Kext

Link State

Link State

Link State

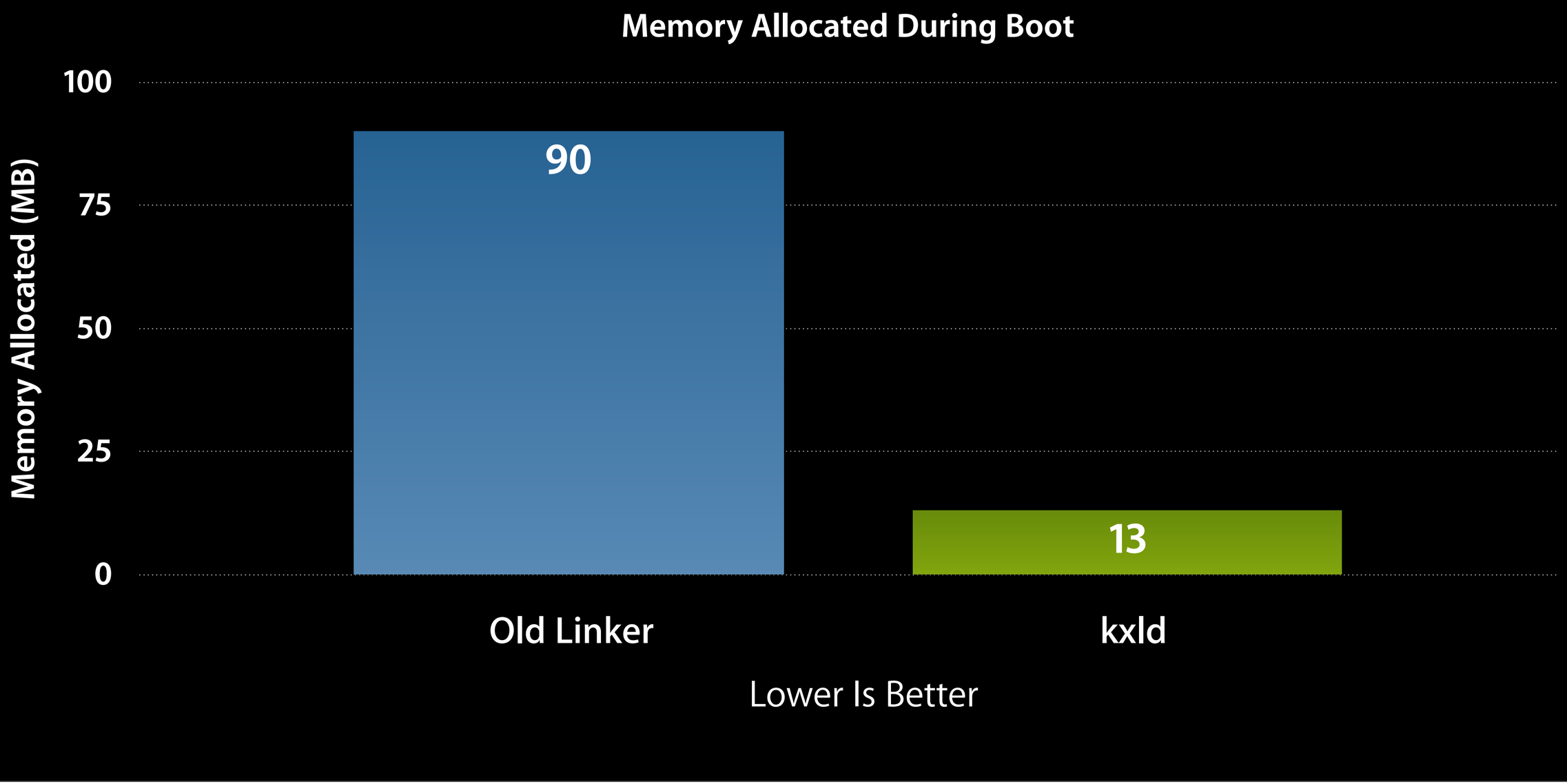
Link State

Link State

Kernel

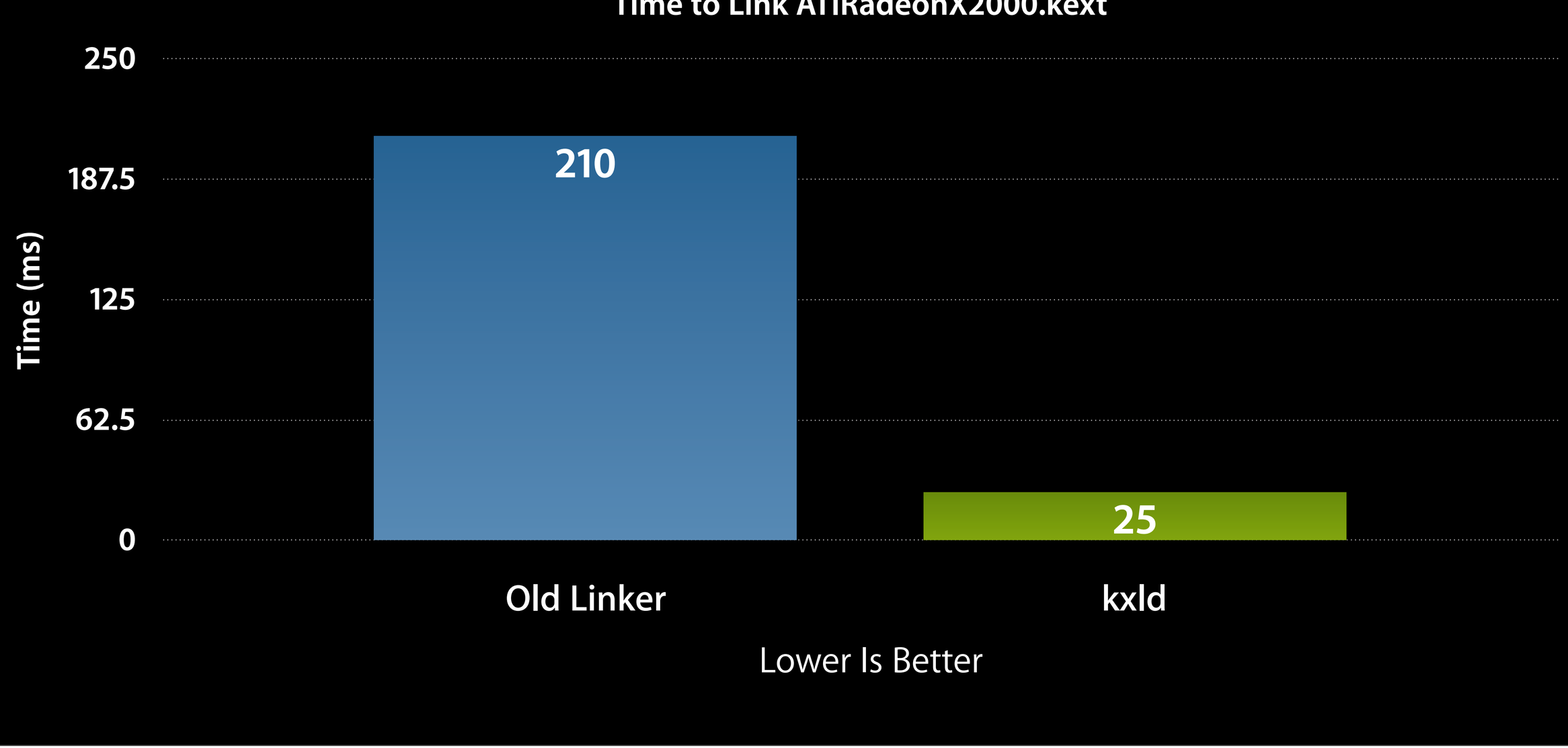
Performance: Memory

Mac Pro (Early 2008)



Performance: Speed

Mac Pro (Early 2008)



Lower Is Better

Features Enabled by kxld

- Cross-architecture symbol generation
- Weak linking for C symbols (K64)
- Segment protection (K64)
  - Write-protected __TEXT
  - Execute-protected __DATA

Kext Runtime Environment

Programming considerations and kext system interfaces

The Kernel Environment

- Longest-running task (do not leak)
- Allocated memory is wired (not pageable)
- Limited stack space
  - Watch your local variable declarations
  - Limit recursion
- Restricted floating-point or vector facilities
- Restricted file I/O
- Incomplete C library support
- Complex debugging

Kexts in the Kernel

- Kexts run within the kernel's 32- or 64-bit virtual address space
- No sandboxing or address space protection
- Errors bring down the whole OS (kernel panic)
- All code running within the kernel is inherently trusted

- If you can implement your feature in user space, do so

- All log messages from the kernel now go to /var/log/kernel.log

Kext Runtime

#include <libkern/OSKextLib.h>

- Kext introspection
- Libkern C++ in generic kexts
- Opt-in auto-unload for generic kexts
- Weak linking for C symbols
- Kext resource requests
- Kext load requests

Kext Introspection

#include <libkern/OSKextLib.h>

```
const char * OSKextGetCurrentIdentifier(void);
const char * OSKextGetCurrentVersionString(void);
OSKextLoadTag OSKextGetCurrentLoadTag(void);
```

- Primarily useful for logging
- Use instead of deprecated kmod_info struct
  - kmod_… functions deprecated or removed

Libkern C++ in Generic Kexts

- Generic kexts are non-I/O Kit kexts
- Libkern C++ is now safe to use in Snow Leopard
- Not supported prior to Snow Leopard
  - Be sure to set OSBundleLibraries versions appropriately!

Opt-in Auto-unload for Generic Kexts

#include <libkern/OSKextLib.h>

```
OSReturn OSKextRetainKextWithLoadTag(OSKextLoadTag tag);
```

- Increments kext reference count
- Enables auto-unload

```
OSReturn OSKextReleaseKextWithLoadTag(OSKextLoadTag tag);
```

- Decrements kext reference count
- Schedules auto-unload scan

K64: Weak Linking for C Symbols

#include <libkern/OSKextLib.h>

`OSKextSymbolIsResolved(symbol)`

- Result is true if `symbol` is valid

Kext Resource Requests

- Request a resource from a kext's bundle
- Useful for briefly needed/optional resources
  - Device firmware
  - Lookup tables
  - State machine data
- Resources not localizable for kernel use
- Resources not available before kextd starts

Kext Resource Request

#include <libkern/OSKextLib.h>

```
OSReturn OSKextRequestResource(
  const char * kextIdentifier,
  const char * resourceName,
  OSKextRequestResourceCallback callback,
  void * context,
  OSKextRequestTag * requestTagOut);
```

- Asynchronous request to kextd

Kext Resource Callback

#include <libkern/OSKextLib.h>

```
typedef void (*OSKextRequestResourceCallback)(
  OSKextRequestTag * requestTag,
  OSReturn result,
  const void * resourceData,
  uint32_t resourceDataLength,
  void * context);
```

- Guaranteed to be invoked unless request canceled
- Kext does not own the invoked thread
- Destroys `resourceData` on callback return

Canceling a Kext Resource Request

#include <libkern/OSKextLib.h>

```
OSReturn OSKextCancelRequest(
  OSKextRequestTag requestTag,
  void ** contextOut);
```

- Cancels callback for tag
- Returns original context for cleanup

Kext Load Request

#include <libkern/OSKextLib.h>

```
OSReturn OSKextLoadKextWithIdentifier(const char * kextIdentifier);
```

- Asynchronous; no callback
- Kexts must provide own rendezvous mechanism

Installing and Uninstalling Kexts

Kext Installation Is Brain Surgery

- Kext installation affects running system
- Kext system guards against gross failures (panics)
- Poor installation yields bad user experience
  - Silent failure of new device
  - Incorrect driver used for new device
  - Alert about improperly installed kext

Installing Drivers

- Devices should not be attached before install
  - Existing drivers will match immediately
  - Matching will happen during install
- Drivers go into /System/Library/Extensions/
- /Library/Extensions/ is reserved for future use
- No symlinks
  - Booter can’t resolve
  - Other hard-to-diagnose problems

Installing Generic Kexts

- Generic kexts can be installed anywhere
  - /System/Library/Extensions/—recommended
  - Application bundle
  - Application support folder
- /Library/Extensions/ is reserved for future use
- *Not* in user folders

Installing Kext Files

Releases prior to Snow Leopard

- Require admin access
- Install (or uninstall) whole kext atomically

- Write to temporary folder on destination volume
- Use postflight script
  - Set owner, group, permissions

```
chown -R root:wheel kexts
chmod -R go-w kexts
```

  - mv into destination folder

Resetting the Kext System

Releases prior to Snow Leopard

- Invalidate kext caches: `touch dest_folders`
- Trigger rescan of extensions folder and restart driver matching
  - Mac OS X 10.0–10.2: Require restart
  - Mac OS X 10.3 and later: `killall -HUP kextd`
  - (Mac OS X 10.5 and later: Automatic, with small delay)
- Load generic kexts
- User can now attach device

- Technical Q&A QA1319: Installing an I/O Kit KEXT Without Rebooting

Installer.app Changes in Snow Leopard

Products supporting only Snow Leopard or later

- Performs atomic install of all files
- Invalidates kext caches following postflight script
- Does not send kextd a SIGHUP

- To force immediate driver matching:

```
touch dest_folders
killall -HUP kextd
```

Kext Caches

- Cache formats and locations have changed with every release
  - And they will continue to change!
- Make no assumptions about caches
- Touch install destination folder(s)
- If creating a bootable DVD, contact Developer Support

Related Sessions and Labs

Creating I/O Kit Drivers for Multiple Architectures and OS Versions	North Beach Tuesday 3:30PM
USB, FireWire, and Bluetooth Lab	OS X Lab C Tuesday 9:00AM–3:15PM
USB and FireWire Lab	Location and time varies Wednesday, Thursday, Friday
64-Bit Kernel Lab	Snow Leopard Lab Wednesday 12:00PM

Summary

- New developer tool: kextutil
- 64-bit kexts
  - Cross-architecture tools support
  - Architecture-specific properties
- Improved kext runtime
  - Programmatic kext loading
  - Resource requests
- Backward compatible

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09